



caBIG

*cancer Biomedical
Informatics Grid*



Introduction to Unified Modeling Language (UML)

*NCICB Software Development Processes
Facilitating Systems Interoperability*

Sashi Thangaraj (SAIC)

Agenda

- ▶ Introduction
- ▶ What is UML?
- ▶ Benefits of UML
- ▶ UML Artifacts
- ▶ Tools
- ▶ Case Study - caBIO
- ▶ Q&A

Introduction

As the world becomes more complex, the computer-based systems that inhabit the world must also increase in complexity. These systems often involve multiple components:

- ▶ Hardware
- ▶ Software
- ▶ Networked across great distances
- ▶ Databases

Qn. If you want to make systems that deal with real world problems, how do you get your hands around real world complexities?

Ans. The key is to organize the design process in a way that clients, analysts, programmers and others involved in system development can understand and agree on. UML is key in providing this organization

What is UML?

- ▶ The Unified Modeling Language (UML) is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of a software system
- ▶ UML was released in 1997 as a method to diagram software design and was designed by a consortium of the best minds in object-oriented analysis and design
- ▶ UML is by far the most exciting thing to happen to the software industry in recent years as every other engineering discipline has a standard method of documentation
 - ▶ Electronic engineers have schematic diagrams
 - ▶ Architects and mechanical engineers have blueprints and mechanical diagrams
 - ▶ The software industry now has UML!

Benefits of UML

- ▶ Software systems are professionally designed and documented before they are coded so that all stakeholders know exactly what they are getting, in advance
- ▶ Since system design comes first, UML enables re-usable code to be easily identified and coded with the highest efficiency, thus reducing software development costs
- ▶ UML enables logic 'holes' to be spotted in design drawings so that software will behave as expected
- ▶ The overall system design described in UML will dictate the way the software is developed so that the right decisions are made early on in the process. Again, this reduces software development costs by eliminating re-work in all areas of the life cycle.

More Benefits...

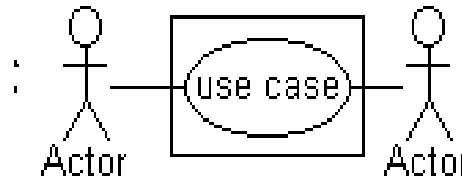
- ▶ UML provides an enterprise level view of the system and, as a result, more memory and processor efficient systems can be designed
- ▶ UML enables ease of maintenance by providing more effective visual representations of the system. Consequently, maintenance costs are reduced.
- ▶ UML diagrams assist in providing efficient training to new members of the development team member
- ▶ UML provides a vehicle of communication with both internal and external stakeholders as it documents the system much more efficiently

UML Artifacts

The following are the most commonly used UML artifacts:

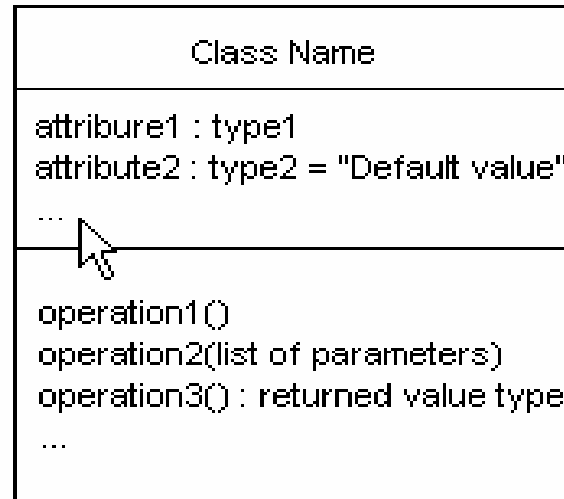
- ▶ Use Case Diagrams
- ▶ Class Diagrams
- ▶ Sequence Diagrams
- ▶ Collaboration Diagrams
- ▶ Activity Diagrams
- ▶ Component Diagrams
- ▶ Deployment Diagrams

Use Cases



- ▶ An ellipse represents a *use case* and a “stick figure” represents an *actor* operating within the *use case*
 - ▶ An *actor* can be a user, system, or other entity
- ▶ The initiating *actor* appears on the left of the *use case*, and the receiving *actor* appears on the right
- ▶ The actor is identified by a name below the “stick figure”
- ▶ The name of the *use case* appears either inside or below the ellipse
- ▶ An *association* line is a line that connects an *actor* to the *use case*, and represents communication between the *actor* and the *use case*
 - ▶ The *association* line is solid, similar to the line that connects associated classes

Class Diagrams



- ▶ The rectangle represents the *class*. The name of the class by convention begins with an initial uppercase letter
- ▶ A one-word *attribute* name is written in lowercase
- ▶ An *operation* name is also written in lowercase

Class Diagrams - continued...

► Associations

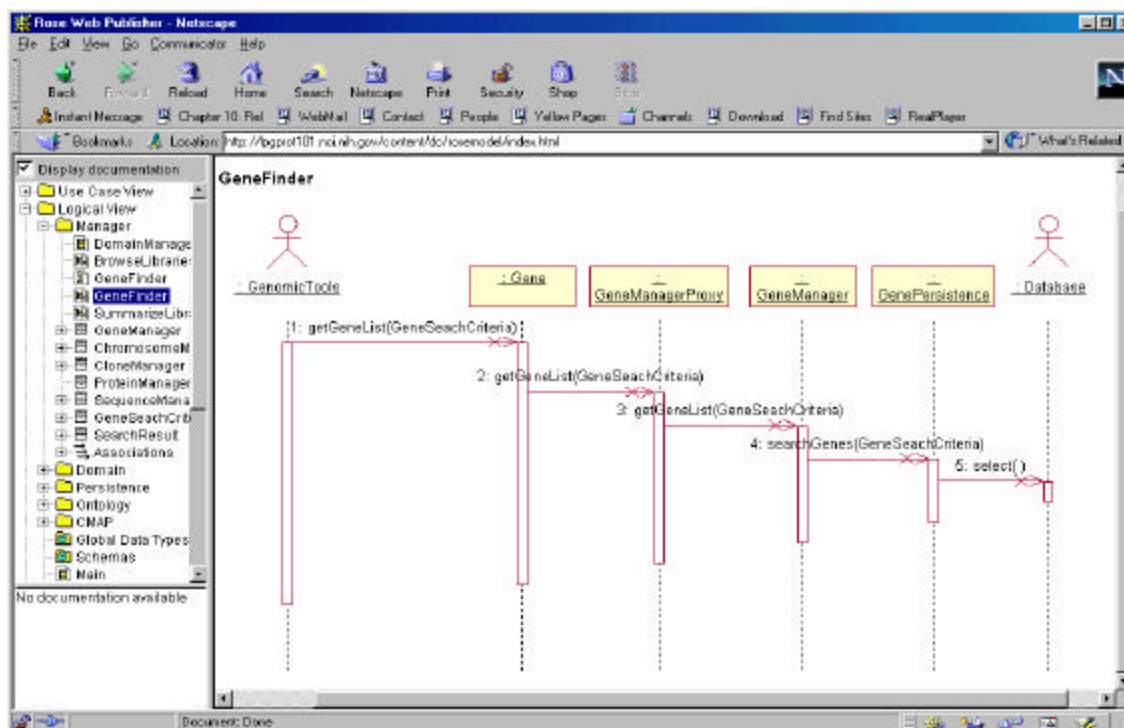
- When *classes* are connected together conceptually, that connection is called an *association*
- Multiplicity is a special type of *association* which shows the number of objects from one *class* that relate to a number of objects in an associated *class*. One *class* can be related to another in the following ways:
 - _ one-to-one
 - _ one-to-many
 - _ one-to-one or more
 - _ one-to-zero or one
 - _ one-to-a bounded interval (one-to-two through twenty)
 - _ one-to-exactly n
 - _ one-to-a set of choices (one-to-five or eight)

Class Diagrams - continued...

- ▶ Inheritance
 - ▶ One *class* (the child *class* or *subclass*) can inherit attributes and operations from another (the parent *class* or *superclass*). The parent *class* is more general than the child *class*.
- ▶ Generalization
 - ▶ In generalization, a *child* is substitutable for a parent. That is, anywhere the parent appears, the child may appear. The reverse isn't true, however.

Sequence Diagrams

- ▶ Generic sequence diagrams often provide opportunities to represent *if statements* and *while loops*
 - ▶ Each condition for an *if statement* is enclosed in square brackets
 - ▶ The condition that satisfies a *while loop* is also enclosed in square brackets and the left bracket is prefixed with an asterisk



Component Diagrams



- ▶ A component diagram contains *components*, *interfaces* and *relationships*
- ▶ The component diagram's main icon is a rectangle that has two rectangles overlaid on its left side. The *component* name appears inside the icon. If the *component* is a member of a *package*, you can prefix the *component's* name with the name of the *package*.

Tools

- ▶ There are a variety of tools that are used to analyze, design, implement, and maintain systems that are designed using UML including:
 - ▶ Design Tools
 - ▶ Implementation Tools
 - ▶ Interactive Development Environments (IDEs)

Design Tools

- ▶ A variety of Computer Aided Software Engineering (CASE) tools that conform to modeling language standards such as UML are available
 - ▶ Enterprise Architect (EA) is a popular CASE tool suite
 - EA is an object oriented tool supporting full life-cycle development
 - Enterprise Architect is a flexible, complete, and powerful UML modeling tool
 - EA facilitates the system development, project management, and business analysis process

Code Generation Toolkits

- ▶ There are a variety of open source projects that read the meta model format (XMI) of a model and generate the code including logic using template languages. Example projects include:
 - ▶ Eclipse
 - ▶ MDA Beans
 - ▶ JET / FML
 - ▶ Other Open Source Modules

Interactive Development Environments (IDEs)

- ▶ There are a varieties of IDEs available to assist in the software development
 - ▶ Eclipse (www.eclipse.org) is a popular universal tool framework that can be leveraged as an IDE for development environment
 - ▶ Eclipse facilitates a plug-in development environment (<http://eclipse.org/pde/index.html>)
 - XML Buddy
 - JET / EMF
 - JSP editor

Case Study - caBIO

- ▶ Analysis
- ▶ Design
- ▶ Implementation

Analysis

► Use Case Document

Find Gene(s) for a given search criteria (keyword)

Usecase ID:100300

Actor

- caBIO Application developer

Starting Condition

The actor establishes reference to the caBIO software

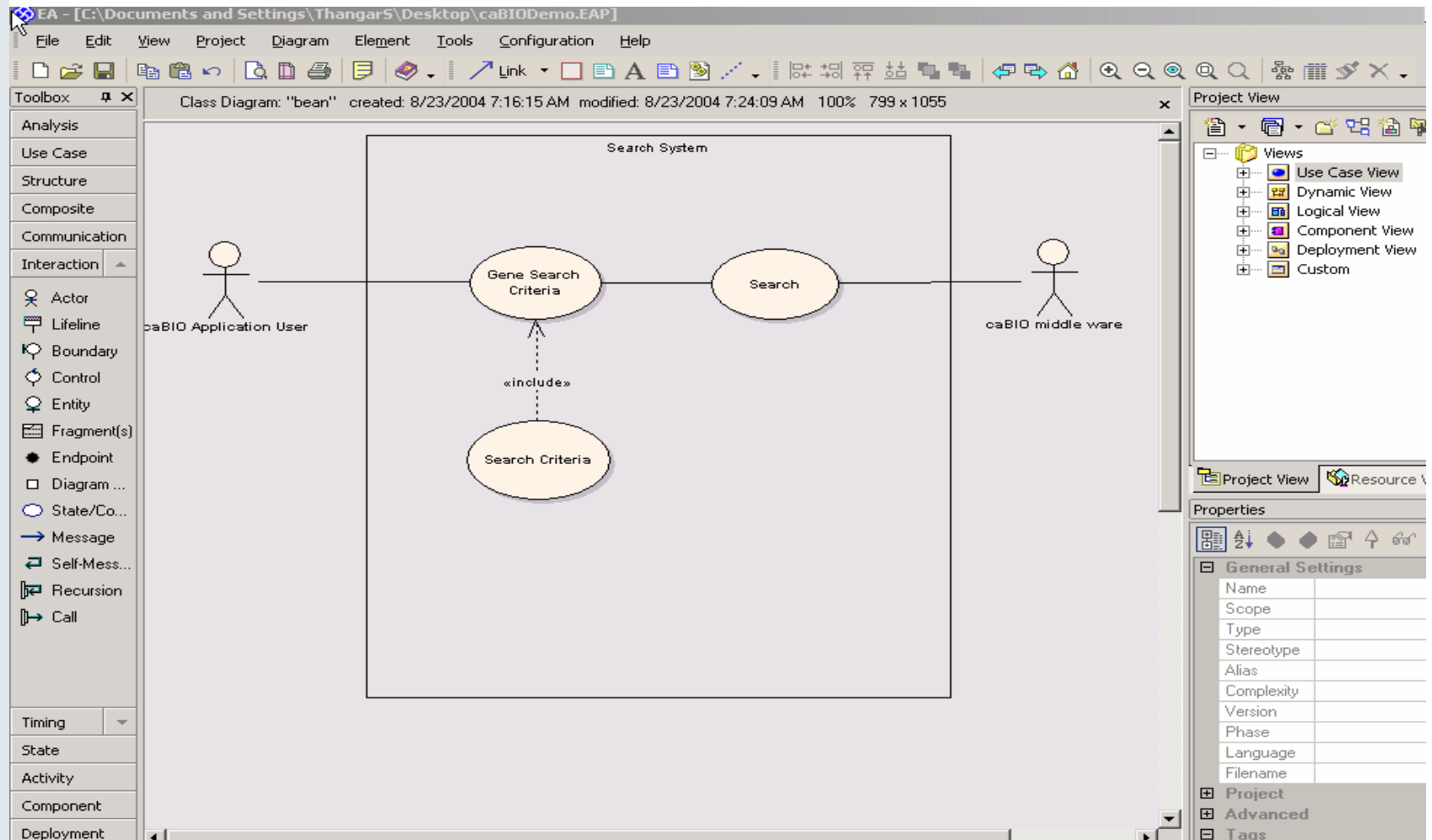
Flow of Events

1. The actor sets the search criteria (Use case ID 101300) using one or more keywords in the criteria
2. Invoke the search use case (Use case ID 105300) and pass the search criteria instantiated at step 1.
3. A result set (Use case ID 110300) is returned to the actor.

End Condition

The actor has obtained a collection of Genes needed for his application.

Creating Use Case Diagram in Enterprise Architect



caBIO Gene Class

```
<<gov.nih.nci.caBIO>>
Gene
(from bean)

dbCrossRefs : Hashtable
bcdVals[] : Logical View::java::lang::String
serialVersionUID : long = 1234567890L
allAssociationMethods : ArrayList = null
associationMethods : Hashtable = null
id : Logical View::java::lang::Long
locusLinkId : Logical View::java::lang::String
OMIMId : Logical View::java::lang::String
title : Logical View::java::lang::String
name : Logical View::java::lang::String
symbol : Logical View::java::lang::String
locusLinkSummary : Logical View::java::lang::String
clusterId : Logical View::java::lang::Long
_bcdsCount : int = - 1
expressionFeaturesCount : int = - 1
cmapOntologiesCount : int = - 1
goOntologiesCount : int = - 1
sequencesCount : int = - 1
snpsCount : int = - 1
_keywordsCount : int = - 1
pathwaysCount : int = - 1
mapLocationsCount : int = - 1
proteinsCount : int = - 1
geneHomologsCount : int = - 1
expressedInOrgansCount : int = - 1
geneAliasesCount : int = - 1
targetsCount : int = - 1
expressionMeasurementsCount : int = - 1
librariesCount : int = - 1
```






























Connecting CDE (Class) to EVS

The screenshot displays the Enterprise Architect (EA) interface. The main window shows a 'Class Diagram: "Logical Model"' with a toolbar and a left-hand toolbox. A 'Class : Gene' dialog box is open, showing the 'General' tab. The 'Tag' field is set to 'CUI' and the 'Value' field is set to 'C1234456'. A callout points to the 'Tag' field with the label 'EVS Concept Unique ID'. Another callout points to the 'Value' field with the label 'EVS CUI Value'. The 'Defined Values' table is visible, showing a property 'persistence' with a value 'transient'. The 'Project View' on the right shows a hierarchy of models, including 'Logical Model' and 'caBIO'. The 'Properties' panel at the bottom right shows 'Class Settings' for the 'Gene' class, including 'Name', 'Scope', 'Type', 'Stereotype', 'Alias', 'Complexity', 'Version', 'Phase', 'Language', and 'Filename'.

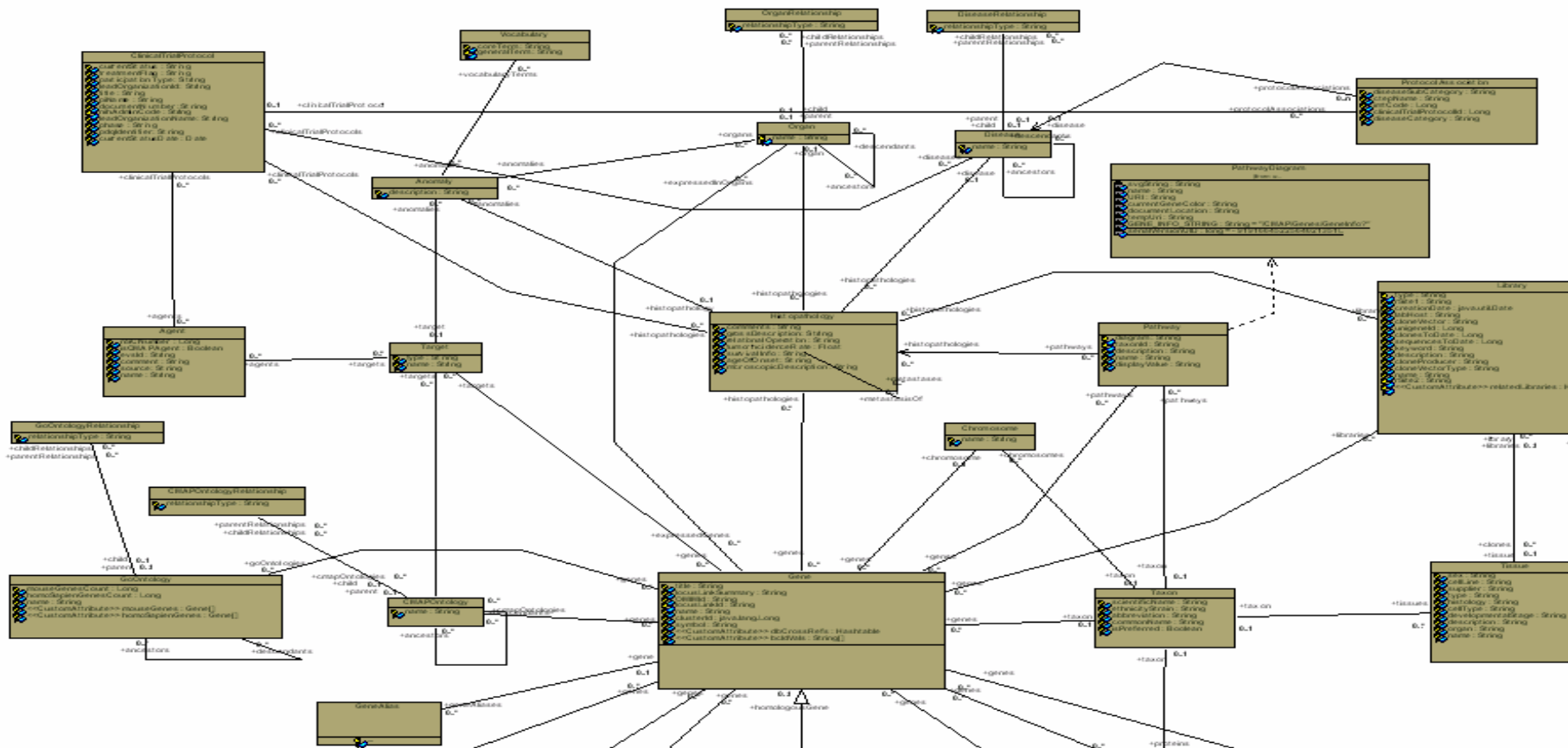
Property	Value
persistence	transient

Class Settings	
Name	Gene
Scope	Public
Type	Class
Stereotype	
Alias	
Complexity	Easy
Version	
Phase	
Language	
Filename	

Gene Class

<<gov.nih.nci.caBIO>>	
Gene	
(from bean)	
	dbCrossRefs : Hashtable
	bcdVals[] : Logical View:java:lang:String
	serialVersionUID : long = 1234567890L
	allAssociationMethods : ArrayList = null
	associationMethods : Hashtable = null
	id : Logical View:java:lang:Long
	locusLinkId : Logical View:java:lang:String
	OMIMId : Logical View:java:lang:String
	title : Logical View:java:lang:String
	name : Logical View:java:lang:String
	symbol : Logical View:java:lang:String
	locusLinkSummary : Logical View:java:lang:String
	clusterId : Logical View:java:lang:Long
	_bcdsCount : int = - 1
	expressionFeaturesCount : int = - 1
	cmapOntologiesCount : int = - 1
	goOntologiesCount : int = - 1
	sequencesCount : int = - 1
	snpsCount : int = - 1
	_keywordsCount : int = - 1
	pathwaysCount : int = - 1
	mapLocationsCount : int = - 1
	proteinsCount : int = - 1
	geneHomologsCount : int = - 1
	expressedInOrgansCount : int = - 1
	geneAliasesCount : int = - 1
	targetsCount : int = - 1
	expressionMeasurementsCount : int = - 1
	librariesCount : int = - 1

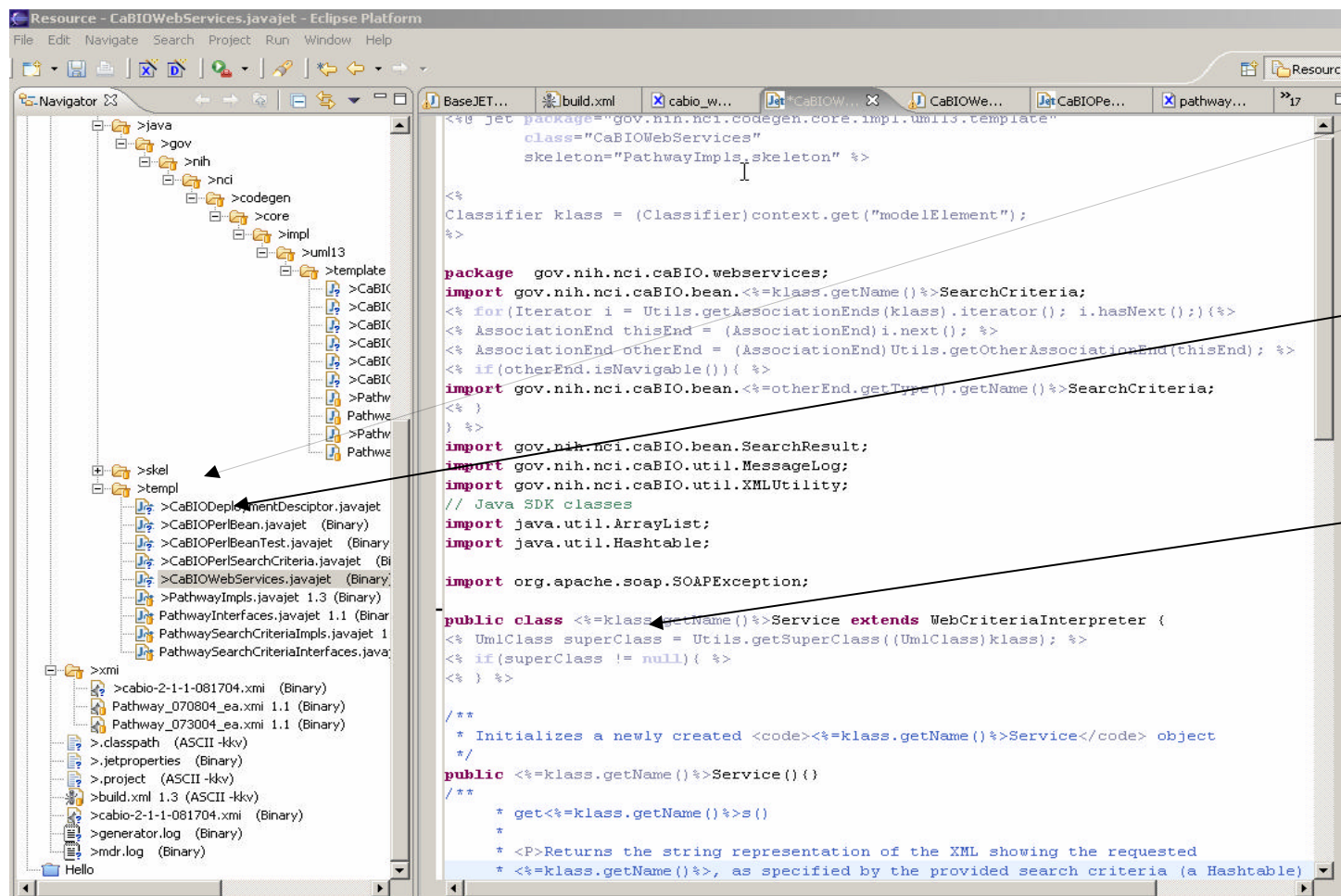
caBIO Object Model



Implementation

- ▶ Code generation using templates
 - ▶ Domain objects in templates
 - ▶ Data model in templates
- ▶ Implement middleware specific managers

Code Generation



The screenshot displays the Eclipse IDE interface. The left-hand pane shows a project tree with a 'template' folder containing 'CaBIOWebServices.javajet'. The right-hand pane shows the generated Java code for 'CaBIOWebServices'.

```
<%@ jet package="gov.nih.nci.codegen.core.impl.Uml13.template"
class="CaBIOWebServices"
skeleton="PathwayImpls.skeleton" %>

Classifier klass = (Classifier)context.get("modelElement");
%>

package gov.nih.nci.caBIO.webservices;
import gov.nih.nci.caBIO.bean.<%=klass.getName()%>SearchCriteria;
<% for (Iterator i = Utils.getAssociationEnds(klass).iterator(); i.hasNext(); ) { %>
<% AssociationEnd thisEnd = (AssociationEnd)i.next(); %>
<% AssociationEnd otherEnd = (AssociationEnd)Utils.getOtherAssociationEnd(thisEnd); %>
<% if (otherEnd.isNavigable()) { %>
import gov.nih.nci.caBIO.bean.<%=otherEnd.getType().getName()%>SearchCriteria;
<% } %>
import gov.nih.nci.caBIO.bean.SearchResult;
import gov.nih.nci.caBIO.util.MessageLog;
import gov.nih.nci.caBIO.util.XMLUtility;
// Java SDK classes
import java.util.ArrayList;
import java.util.Hashtable;

import org.apache.soap.SOAPException;

public class <%=klass.getName()%>Service extends WebCriteriaInterpreter {
<% UmlClass superClass = Utils.getSuperClass((UmlClass)klass); %>
<% if (superClass != null) { %>
<% } %>

/**
 * Initializes a newly created <code><%=klass.getName()%>Service</code> object
 */
public <%=klass.getName()%>Service() {}

/**
 * get<%=klass.getName()%>s()
 *
 * <P>Returns the string representation of the XML showing the requested
 * <%=klass.getName()%>, as specified by the provided search criteria (a Hashtable)
```



caBIG

*cancer Biomedical
Informatics Grid*



Q & A

- ▶ <http://ncicb.nci.nih.gov/core/caBIO>